Rhythm Interplant Manual

Interplant:
Multi-Plant Manufacturing

Rhythm 2.5

Rhythm Intelligent Planning and Scheduling Systems Interplant Manual

Rhythm Interplant Version 2_5

i2 TECHNOLOGIES, INC.
(Formerly Intellection, Inc.)

April 22, 1994

The information and/or drawings set forth in this document are undergoing continuous improvement. Cosmetic items are being enhanced to provide better visual appearance. Text is being reworded and added to improve the clarity of the information and to increase the ease of finding the information required.

If you have any comments or suggestions concerning this document, please write them and hand to your i2 Technologies representative, or mail to the address provided.

COMMENTS _____

Rhythm Interplant Manual
c/o i2 Technologies, Inc.
1603 LBJ Freeway
Suite 780
Dallas, Texas 75234

Telephone: (214) 620-0026

Table of Contents

List of Figures

List of Tables

(Documentation under development)

Introduction

Section 1 Introduction

1.1 Contents of this Document

The material in this manual is organized into the following sections and appendices:

Section 1: Introduction — Explains the purpose and content of this manual.

Section 2: User Customization — Describes how to customize *Rhythm™* Interplant.

Section 3: Data Requirements — Provides step by step instructions for preparing data files and running Interplant.

Appendix A: Deployment Without rhythm_client Executables— Summarizes how to deploy Interplant when any or all of the plants are not connected on a network.

1.2 Statement of Definition

Rhythm™ Interplant is a module available as an add-on to *Rhythm™* MPPS. It provides the ability to link together multiple copies of *Rhythm™* MPPS, each running independently in separate *plants* of the same company. The linkages allow one plant to procure parts from or supply parts to any other plant in the network:

- Links multiple *rhythm_servers* having different data models in a demand/supply network of *plants*
- Demand: Plant1 requests part, quantity, and due date from Plant2
- Supply: Plant2 responds with quantity and promise date, considering material and capacity constraints
- Adjustment: Plant1 adjusts plans if response is short of request

Interplant can also be used to connect plants of different companies having supplier, customer relationships. This connection can enhance planning and scheduling performance because the plans of the various companies become synchronized.

1.3 Purpose

The Interplant executable propagates demand and supply data as quickly as servers can iterate, and keeps data transfers independent of the *rhythm_server* by means of a separate executable (*rhythm_interplant*). By doing so, it satisfies these needs:

- MPPS servers run in parallel during the day or shift, then save their Interplant demands on each other. On the next run, each of the servers reads demands placed on it, plans them, and issues responses and new Interplant demands. On the third and

successive runs, they comprehend responses and do the same things done during the second run.

- An MPPS server receiving an Interplant demand fills that demand with manufacturing orders, inventory, and procurements from vendors or other MPPS servers. The latter case allows a multilevel demand/supply chain where Plant1 places a demand on Plant2, whose plan for it places a demand on Plant3. Plant2 responds to Plant1, and Plant3 responds to Plant2.
- When an MPPS server has issued an Interplant demand but has not yet received a response, it assumes a fixed lead time (just as with standard vendors) until the response comes back. Thus, the time to make tentative plans for parts requiring Interplant demands is only one day or shift (run of *Rhythm™*). However, the time to finalize plans is:

  *{depth of interplant demand/supply chain - 1} * 2 days/shifts* (runs of *Rhythm™*).

1.4 Procedure

Each *rhythm_server* has an accompanying *rhythm_interplant* running to coordinate the processing. Each server iterates through the following set of steps up to the maximum specified number of iterations (*max_iterations*; See User Customization, Section 2):

1. Process demand and supply data from other plants
2. Run *CAO™*
3. *Save Plan*
4. Send demand and supply data to other plants
5. Wait for other plants to return new data

1.5 Terminology

The following is an alphabetical list of terms used in this document. These terms are included so there is no confusion as to commonly used words as well as a reference to less common words.

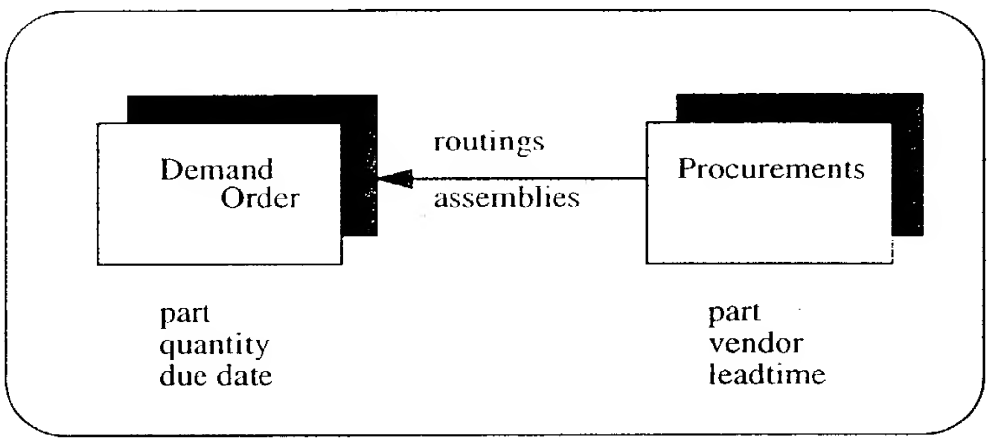
1.5.1 CAO

CAO™ refers to the part of the *rhythm_server* which plans order start times within machine capacity.

1.5.2 Demand Order

See FIGURE 1.

FIGURE 1 Interplant - Demand Order



1.5.3 rhythm_server

rhythm_server refers to the *Rhythm™* MPPS server executable.

1.6 Plants

Within a company, the division of the data model into plants is arbitrary from the standpoint of *Rhythm™ Interplant*. Typical divisions are by factory location, product group, machine group, or some other logical combination. These divisions usually exist before a customer purchases *Rhythm™* MPPS.

The customer has the choice of continuing to plan the plants separately by deploying multiple copies of MPPS linked together with *Rhythm™ Interplant*, versus consolidating the plants (in terms of data and possibly planning personnel) to be planned by one MPPS. Using a single consolidated MPPS results in more tightly coupled plans among the various plants, whereas using Interplant may be desirable if:

- the size of the data model requires distributed processing to achieve desired performance levels
- the customer prefers not to invest in the effort to consolidate multiple plant data into one MPPS data model
- organizational constraints within the company make it difficult to consolidate the planning personnel of multiple plants
- the customer wishes to use Interplant to couple his plans with the plans of some of his vendors or customers

Finally, *Rhythm™ Interplant* can speed deployment of *Rhythm™* MPPS into the plants. Thereafter, a customer may choose to achieve tighter coupling of key plants by replacing all or part of the Interplant network with a consolidated *Rhythm™* MPPS.

1.7 Logic

Each plant treats the other plants in the Interplant network as a special type of part vendor. When a part is needed from another plant, it is initially assumed to be procured at a user defined lead time, which is specified in the data model. However, this assumption is usually inaccurate because the supplier plant may have capacity or material shortages which delay delivery beyond the fixed lead time. Conversely, the supplier plant might be able to deliver the part sooner which is advantageous when the demanding plant needs it before the defined lead time.

Through data exchanges among plants, *Rhythm™ Interplant* converts the lead time assumptions into actual delivery dates which comprehend each plant's capacity and material constraints. The rate of exchange is user controlled and typically depends on various factors such as whether the plants are networked, whether *Rhythm™ Interplant* is done gradually during the day or run through many iterations over night.

1.7.1 Plant-Level Demand Logic

Each plant's *rhythm_server* iterates through the following steps:

1. Procures parts using both normal and Interplant vendors. Assumes fixed lead times.
2. Sends Interplant procurements to other plants: part, quantity, and time preferred.
3. Waits for other plants to send back responses: quantity and promise date.
4. Adjusts plans if necessary:
   - If quantity is short, build or procure the balance.
   - If promise date is late, leave the order late.
   - If promise date is sooner than lead time, take no action.

1.7.2 Plant-Level Supply Logic

Each plant's *rhythm_server* iterates through the following steps:

1. Reads other plant procurements sent to this one and generates *Interplant demand orders*.
2. Plans the orders as normal:
   - May include procurements from other plants
   - Includes capacity constraints depending on the command line option *run_cao*
3. Sends responses:
   - quantity = output quantity of order's final assembly
   - promise date = planned completion time of final assembly

1.8 Plan Quality

The quality of the plan depends on the following items:

- Good Interplant lead time estimates
- Few alterations to Interplant demand order plans

- Simplicity of the vendor data model
- Propagation of demand and supply data quick enough for:
  - depth of supply chain
  - shortness of manufacturing cycle times
  - length of planning cycle (*rhythm_server* run frequency)

1.9 Benefits/Limitations

Rhythm™ Interplant provides trade-offs between the following benefits and limitations.
Benefits:

- Links different companies, planning groups, or data models
- Deploys *Rhythm™* MPPS incrementally
- Achieves quicker run times on large data sets

Limitations are a result of limited visibility into other plants:

- by users when viewing and manipulating plans
- by *Rhythm™* when generating plans

1.10 Hardware Requirements

Ideally, each plant's *Rhythm™* MPPS program has disk or network access which allows it to send data files to every other plant's *Rhythm™* data directory. These files communicate the part demand and supply information among the plants.

In cases where disk or network access is not possible, it is feasible (but less reliable) to transfer the information between plants by modem or by overnight mail of tapes or floppy disks.

1.11 Data Requirements

Rhythm™ Interplant requires some additional data to be supplied to each plant's *Rhythm™* MPPS:

- the names and data locations for each plant (*supplier_data* file)
- the parts obtainable from each plant and the initial lead times to assume before the other plants can quote actual delivery dates (*supplier_part_data* file)
- part translations in case the plants identify the same part by different names (*supplier_part_data* file)
- control parameters such as how long to wait for data from the other plants. See User Customization, Section 2, for descriptions of command line options.

1.12 Data Files

Entries from the standard specification file apply to *Interplant*. See the *Rhythm™* Data Dictionary for more details. The Interplant_Order_Record is shown:

```
interplant_orders
    mode: readwrite
    Optional: True
    Interplant_Order_Record:
      supplying_order,
      demanding_order,
      operation,
      consumer,
      part;
```

The Interplant_Procurement_Record is shown:

```
interplant_data_PLANTNAME
 # Do not give this a readwrite mode.  The customer spec file must "use:" this
 # in a write mode file with PLANTNAME appropriate for the customer, such
 # as interplant_data_plant1 :use interplant_data_PLANTNAME :mode write;
 # If we add a mode here, Save Plan does not know which file to write.
 #
    Optional: True
    Interplant_Procurement_Record:
      is_demand_p,
      demanding_order,
      operation,
      consumer,
      part,
      time,
      quantity,
      priority,
      supplier;
```

The Supplier_Record is shown:

```
supplier_data
    Optional: True
    Supplier_Record:
      supplier,
      type,
      data_directory;
```

The Supplier_Part_Record is shown:

supplier_part_data
Optional: True,
Supplier_Part_Record:
vendor,
material_type,
lead_time,
lead_time_uom,
max_quantity,
vendor_part;

1.13 Example

The following example (see Table 1:) illustrates an application of Interplant. See FIGURE 2.

Assumptions:

- Plant2 supplies a part P21 to Plant1
- Plant3 supplies a part P32 to Plant2
- Plant2 builds P21 using P32

FIGURE 2 Interplant Example - 3 Plants

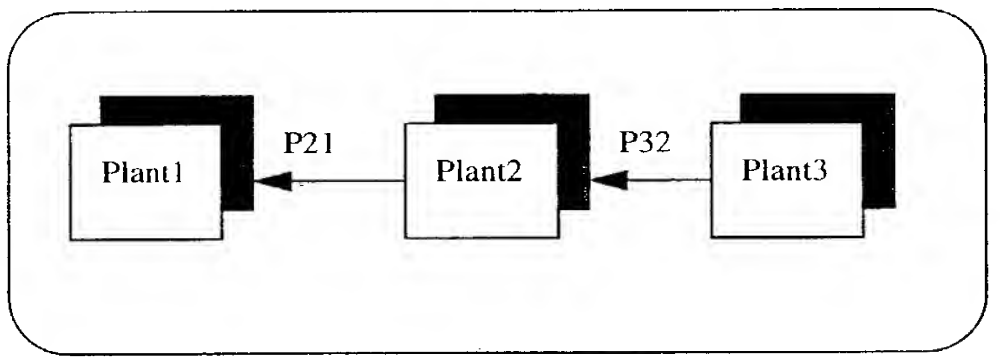


Table 1: Interplant Example

Iteration	Plant Action
1	Plant1 demands P21 from Plant2 and assumes a lead time until iteration #3. Plan quality = OK

Table 1: Interplant Example

Iteration	Plant Action
2	Plant2 plans P21 and sends Plant1 a tentative promise date. It also sends Plant3 a demand for P32, assuming a lead time until iteration #4.
3	Plant1 comprehends the promise date from Plant2, and Plant3 plans its Interplant demand from Plant1. Plan quality = better
4	Plant2 comprehends the promise date from Plant3 and possibly adjusts its plan for the Interplant demand from Plant1.
5	Plant1 comprehends the adjustment to the promise date from Plant2 and adjusts its plans. Plan quality = best

1.14 Project Nomenclature

CAO™	Constraint Anchored Optimization
DS	Detailed Scheduling
GUI	Graphical User Interface
MPPS	Master Production Planning & Scheduling
UI	User Interface

1.15 References

1. APICS Dictionary
2. *Rhythm™* Data Dictionary
3. *Rhythm™* Data Dictionary for the Minimum File Set
4. *Rhythm™* Data File Specification
5. *Rhythm™* Data Files for [customer]
6. The New *Rhythm™* Data Dictionary
7. The UNIX Programming Environment
8. X Window System User's Guide

User Customization

Section 2 User Customization

2.1 Introduction

Rhythm™ is designed to be easily customizable. This capability provides the flexibility desired by customers for customizing *Rhythm™* to the needs of their own manufacturing situation. User customization may be implemented through one of the following mechanisms:

- *server* - *Rhythm™* defaults supplied at the server level
- *client* - *Rhythm™* and application defaults supplied at the client level
- *interplant* - *Rhythm™* and application defaults supplied at the interplant level
- *spec_file* - custom formatting of data files

The Application Defaults (*.ad*) file allows for customization of the behavior of the interplant program.

2.2 Overview

This section begins with a brief overview of each of the customization mechanisms. It then presents a discussion of topics which apply to both the server and client, and to both *Rhythm™* and application defaults:

- Naming Defaults Files
- Specifying Defaults

2.2.1 Server

The *server* program runs with a specific set of defaults. The default values with which the program runs may be modified by supplying:

- *Rhythm™* defaults option/value pairs on the *Rhythm™* server command line
- *Rhythm™* defaults option/value pairs in *.rd* files

2.2.2 Client

The *client* program runs with its own specific set of defaults. The default values with which the program runs may be modified by supplying:

- *Rhythm™* defaults option/value pairs on the *Rhythm™* client command line
- *Rhythm™* defaults option/value pairs in *.rd* files
- application defaults, via X Window Resources, in *.ad* files

2.2.3 Interplant

The *interplant* program runs with its own specific set of defaults. The default values with which the program runs may be modified by supplying:

- *Rhythm™* defaults option/value pairs on the *Rhythm™* interplant command line
- application defaults, via X Window Resources, in *.ad* files

2.2.4 Spec_File

The *spec_file* specifies the contents of each data file to be read. The spec_file can be explained as *how to read the customer's file*. Thus, *Rhythm™* can be customized to read the data in a format that is easy for the customer to provide.

2.3 Naming Defaults Files

See *Rhythm™* User's Manual.

2.4 Specifying Defaults

2.4.1 Command Line Option

Interplant default values may be specified on the command line. For example.

rhythm_interplant -max_iterations 10 -port 6162

where

rhythm_interplant	is the name of the program
-max_iterations	is a Long_Type default set to integer 10
-port	is a Custom_Type default set to integer 6162

If a default on the command line is not recognized, a help message is displayed, and the program terminates. The help message specifies all of the known default names, types, and values.

2.4.2 Default Types

Each default has a name and a data type. The data type is a descriptor for how the value of the default is to be entered on the command line or in the *Rhythm™* Defaults file. The available data types are:

- *Custom_Type* - similar to String_Type, except a function is run, with the string as a parameter, which can convert the string and implement it
- *Flag_Type* - a Boolean flag which has a value of either TRUE or FALSE

  Command line format - As a command line option, this type is implemented by preceding the default with a plus or minus sign to indicate its value:

  -flag set it to a value of TRUE

  +flag set it to a value of FALSE (the default)

Defaults file format - As a line in a *.rd* file, this type is implemented with the name followed by a colon, then by TRUE or FALSE:

flag1:FALSE

flag2:TRUE

- *Float_Type* - the default represents a floating point number
- *Long_Type* - the default represents a long integer
- *String_Type* - the default represents a character string

2.4.3 Runtime Defaults

Some default names are not known until runtime, because the names come from data files, or from internal tables which are not accessible from the default code. A plus or minus should not precede these options. If a sign is entered before one of these default names, the name will not be recognized by the program. No checking is done for misspelled or unknown options.

2.4.4 Defaults File Format

The defaults file format is modeled after X Window resource files. There is one line per default. Each line has the default name followed by a colon, then by a value. For Booleans, the value will be either TRUE or FALSE:

name: value

☞ A space may follow the colon but not precede it.

2.5 Available Interplant Defaults

To view a list of the available *Rhythm™* Interplant defaults which may be used as command line options, the *Rhythm™* Interplant executable should be run with a -help option. The output resulting from the -help option is shown in tabular form in Table 2:. A description of each of the options follows the table. Options followed by (*) are for use by developers only.

Table 2: Available Interplant Defaults

Option	Type	Value
-exit_requests	String_Type	""
-max_iterations	Long_Type	20
-num_scans_during_patience	Float_Type	30
-patience	Float_Type	30
-port	Custom_Type	

Table 2: Available Interplant Defaults

Option	Type	Value
+replan	Flag_Type	-
-run_cao	Flag_Type	+
-server	String_Type	(NULL)

-exit_requests Specifies requests to send to the server before exiting Interplant. One option is *write_interplant_report_request*.

-max_iterations Specifies the number of iterations to run. Default value is 20.

-num_scans_during_patience Specifies how often to look for data from other plants during *-patience*. Default value is 30.

-patience Specifies the maximum number of minutes to wait between iterations for data from other plants. Default value is 30.

-port The client and server communicate over TCP/IP sockets. *port* specifies the port number used for the client and server, a unique address for the server process. To run more than one server on a machine, a different port number is specified on the command line for each execution of the *Rhythm™* server. To run Interplant on any server on the machine, the port of the server of interest is specified on the command line.

-replan Does the Interplant processing, *CAO™*, and save_plan steps before sending data to other plants, on each iteration.

-run_cao Runs *CAO™* when *-replan* is TRUE.

-server <hostname> Requests connection of Interplant to the machine on which the server is running. The default is the same machine from which Interplant is run.

3 *Data Requirements*

Section 3 Data Requirements

3.1 Instructions

The following instructions describe required data files and how to deploy *Rhythm™* Interplant at sites already running *Rhythm™* MPPS. Sites are called *plants* in this document and other documents, such as the *Rhythm™* Data Dictionary. Together, the group of plants is called an *Interplant Network*.

1. Add a *supplier_data* file to each plant's data set. The contents of this file will vary for each plant. It provides the plant's name, the names of other plants in the Interplant network, and the paths to the data directories of the other plants. The paths are used by the Interplant executable for transferring data among the plants.

   This file is typically small and easy to construct by hand. Refer to the *Rhythm™* Data Dictionary for details on the format of this file.

   Example:

   supplier_data (for *plant1*)

supplier	type
plant1	*SELF*
plant2	*INTERPLANT*
plant3	*INTERPLANT*

   supplier_data (for *plant2*)

supplier	type
plant1	*INTERPLANT*
plant2	*SELF*
plant3	*INTERPLANT*

   supplier_data (for *plant3*)

supplier	type
plant1	*INTERPLANT*
plant2	*INTERPLANT*
plant3	*SELF*

2. Extend the *supplier_part_data* files to include the parts which can be shipped among plants in the Interplant network. Each plant already has a *supplier_part_data* (or *vendor_data*, the obsolete name) file which specifies the vendors for its parts (usually raw parts). Depending on the customer's current data base, this task may require most of the work of deploying Interplant.

   Refer to the *Rhythm™* Data Dictionary for details on *supplier_part_data*. For each plant, the records to be added to its *supplier_part_data* are the parts which can be supplied (procured) from other plants. The other plants' names (specified in *supplier_data*) appear in the *vendor* field. The *lead_time* field establishes the esti-

mated time to procure the part from that plant. Since this lead time fluctuates with demand and capacity, the customer may wish to regularly alter it by using the file editor (the *Data Files* option in the *Edit* menu of the *Main Window* for *Rhythm™* MPPS) for *supplier_part_data* or by editing the database that populates it.

Plants in an Interplant network will often have different names for the same part. The *vendor_part* field defines the Vendor's name for part_number, and so establishes part name translations among the plants.

The *max_quantity* field should be empty (representing infinity) for Interplant records in order to avoid undesirable behavior.

Example:

supplier_part_data (*vendor_data*) for *plant1*

```
#
# vendor_data / supplier_part_data
#
# Raw Materials
```

vendor	part_number	lead	lead_uom		max_qty	vendor_part
Vendor	*Rubber*	*4*	*WEEKS*	- - -	*1600*	
Vendor	*Plastic*	*4*	*WEEKS*	- - -	*6000*	
Vendor	*CordWire*	*4*	*WEEKS*	- - -	*1600*	
Vendor	*Paper*	*4*	*WEEKS*	- - -	*500*	
Vendor	*Ink*	*4*	*WEEKS*	- - -	*2000*	
Vendor	*Brush*	*4*	*WEEKS*	- - -	*100*	
Vendor	*BrushAxel*	*4*	*WEEKS*	- - -	*100*	
Vendor	*Motor*	*4*	*WEEKS*	- - -	*100*	
Vendor	*Filter*	*4*	*WEEKS*	- - -	*100*	
Vendor	*Switch*	*4*	*WEEKS*	- - -	*100*	
Vendor	*Cardboard*	*4*	*WEEKS*	- - -	*100*	
Vendor	*Print*	*4*	*WEEKS*	- - -	*100*	
Vendor	*PlugContact*	*4*	*WEEKS*	- - -	*100*	
Vendor	*PlugCase*	*4*	*WEEKS*	- - -	*100*	

```
#
# Interplant Parts:
#       BrushCase is named P2BrushCase for plant2 to illustrate
#       the part renaming functionality
```

plant2	*BrushCase*	*1*	*WEEKS*	- - -		*P2BrushCase*
plant3	*BowlStorage*	*1*	*WEEKS*	- - -		
plant3	*BowlCase*	*1*	*WEEKS*	- - -		
plant3	*FilterRim*	*1*	*WEEKS*	- - -		
plant3	*HandleGrip*	*1*	*WEEKS*	- - -		
plant3	*PlugShaft*	*1*	*WEEKS*	- - -		

3. Create a unique spec_file for the data set of each plant. Interplant demand and supply data is written by each *Rhythm™* MPPS server in the Interplant Network whenever the user or the Interplant executable issues a *Save Plan* command. The files written contain the name of the plant (established in *supplier_data*). The file names are of the form

   interplant_data_xxx

   where xxx is the name of the particular plant. For example, when running a server whose *supplier_data* names it Plant1, Save Plan writes out *interplant_data_Plant1*.

   The data set for each plant must include its own unique spec_file which establishes the names of all *interplant_data_xxx* files to be read by that plant and the one *interplant_data_xxx* file to be written by that plant on Save Plan. For example, if Plant1 procures from Plant2 and Plant3, its spec_file should include the following entries (notice the *mode*: field in each entry):

   ```
   # include Rhythm™ standard spec_file

   std_spec_file  spec_file;

   interplant_data_Plant1
   use: interplant_data_PLANTNAME
   mode: write;

   interplant_data_Plant2
   use: interplant_data_PLANTNAME
   mode: read;

   interplant_data_Plant3
   use: interplant_data_PLANTNAME
   mode: read;
   ```

   If Plant2 procures only from Plant3, its spec_file contains:

   ```
   # include Rhythm™ standard spec_file

   std_spec_file  spec_file;

   interplant_data_Plant2
   use: interplant_data_PLANTNAME
   mode: write;

   interplant_data_Plant3
   use: interplant_data_PLANTNAME
   mode: read;
   ```

   These files can not be named arbitrarily. The format must be *interplant_data_PLANTNAME*, and the names must match those in *supplier_data*. Thus, *interplant_data_Plant1* is properly named only if *Plant1* occurs in

supplier_data.

Note that *interplant_data_PLANTNAME* is the generic format for Interplant data files defined in std_spec_file. See the *Rhythm™* Data Dictionary. Since *Rhythm™* maintains this file, it is not important to understand the format in order to deploy Interplant. However, the *Rhythm™* Data Dictionary might help clarify how Interplant works.

It is important to get the file names and modes correct. For example, if Plant1 does not specify *write* mode for *interplant_data_Plant1*, *Save Plan* will not write out its Interplant data.

4. Add the *Interplant Report* to the *Reports* menu by adding the following line to */project/rhythm/customers/CUSTOMERNAME/rhythm_client.ad* where CUSTOMERNAME is the customization directory for the given customer:

```
! Make the Interplant Report visible.
*planner_main_menu_bar*interplant_report_button.wcManaged: TRUE
```

This line may already exist, in which case its value should be set to TRUE.

5. Review and set command line options. The *rhythm_interplant* executable has various command line options for controlling its behavior. It is critical to review each of these options and set them appropriately. Unlike many *rhythm_server* options, the default values for *rhythm_interplant* options are not as generically applicable.

The primary *rhythm_interplant* command line options are (See User Customization, Section 2):

-server
-port
-run_cao
-replan
-max_iterations
-patience
-num_scans_during_patience
-exit_requests

6. Run the *rhythm_interplant* executable. The *Rhythm™* MPPS server is assumed to be already running. See the *Rhythm™ User's Manual* for details on starting *rhythm_server*. *Rhythm™* Interplant releases include an executable called *rhythm_interplant* which resides in the same directory as each plant's *rhythm_server* executable. *rhythm_interplant* is a client much like *rhythm_client*, with the exception of having no windows (no GUI). It connects to the *rhythm_server*, which has the same *-port* number (a command line option for all of these executables).

rhythm_interplant puts the server into a mode of iteratively performing Interplant calculations and transfers of *interplant_data_xxx* files to other plants, which ideally should also be doing likewise. The planning of parts among plants is thus resolved whenever the user chooses to run the *rhythm_interplant* executables. The user ideally should start all of the executables at the same time. The time could be scheduled overnight or at various times throughout the day.

For customers who can not connect by network all of their plant data directories, see *APPENDIX A: Deploying Without rhythm_interplant Executables.*

3.2 Testing

Test the data file specifications as follows:

- Run each plant's *rhythm_server*, then *Save Plan*. Check each plant's server to see that it saves its *interplant_data_xxx* file. If not, a problem exists in its spec_file or *supplier_data* file.
- Without exiting the *rhythm_server* for each plant, run each plant's *rhythm_interplant* with the following command line options:

  rhythm_interplant -port xxx -max_iterations 1

  Check that each server correctly transferred its *interplant_data_xxx* file to the other plant's data directories. If not, the data path fields in *supplier_data* are probably incorrect.
- Exit each rhythm_server and rerun them using the *-progress* command line option to see which files get loaded. Check that the *interplant_data_xxx* files applicable to each plant are read. They are actually read twice. If not, the server's spec_file probably is not set up as specified in the previous step.

3.3 UI Analysis

The following items in the *Rhythm™* client UI relevant to *Rhythm™* Interplant should be noted:

- There are no special *Interplant* tags in the GUI. Interplant data is part of the normal set of orders, vendors, and plans shown by the GUI.
- *Material Register* shows part lead times from other plants
- *Material Register* shows procurements made to other plants
- *Material Register* shows reservations confirmed by other plants
- *Demand Order* related windows include orders which supply other plants (customer field = plant ID)
- *Interplant Report* option in the *Reports* menu of *Main Window.* See FIGURE 3, FIGURE 4, and FIGURE 5 for Interplant reports of three plants from an Interplant network.

FIGURE 3 Interplant Report - Plant1

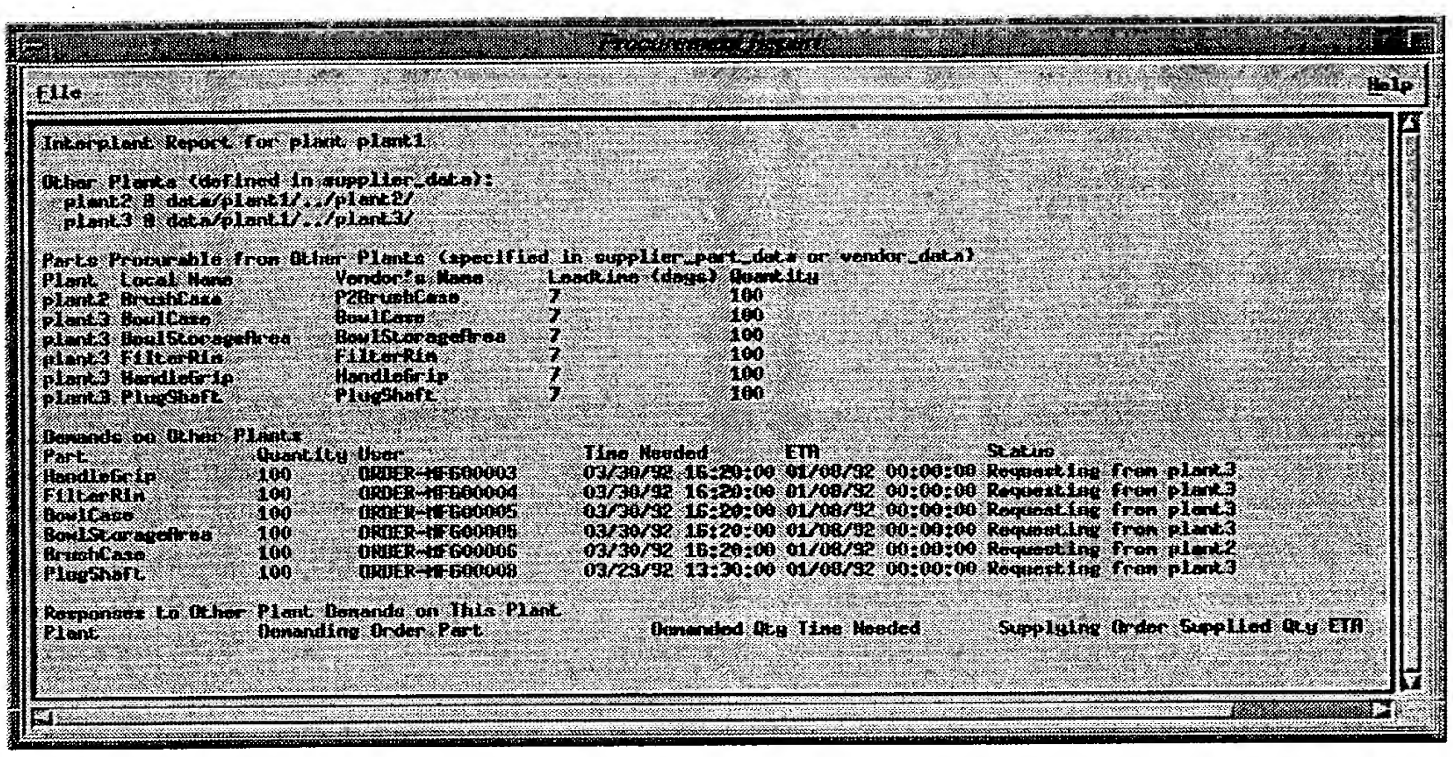


FIGURE 4 Interplant Report - Plant2

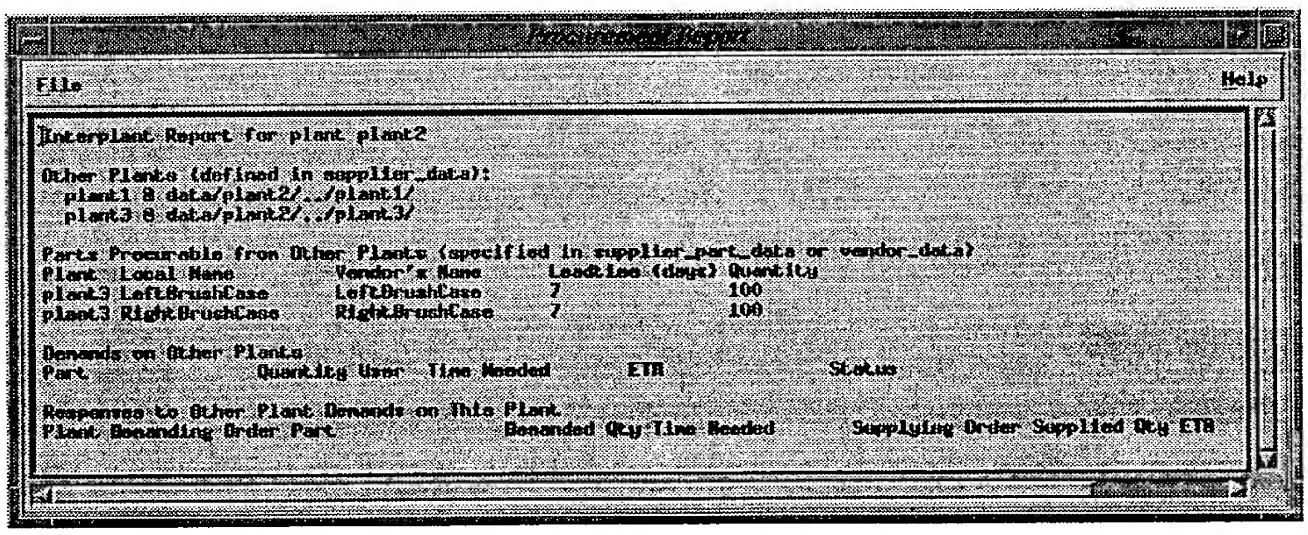
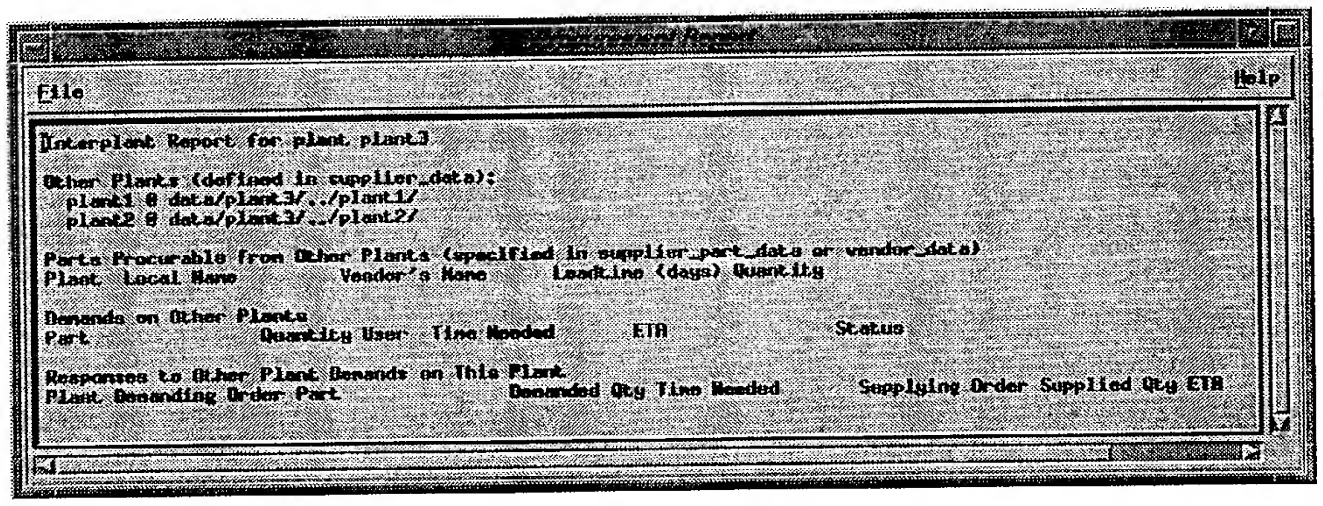
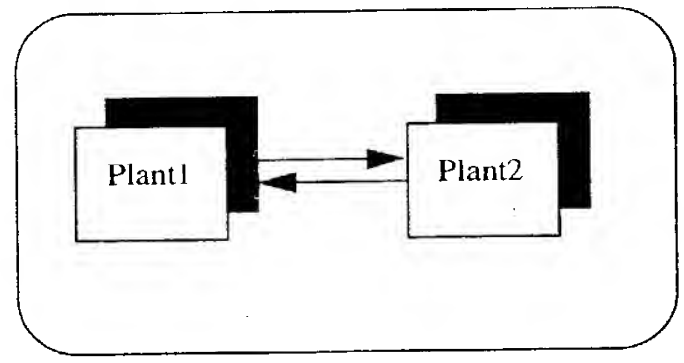


FIGURE 5 Interplant Report - Plant3



3.4 Example

FIGURE 6 Interplant Example - 2 Plants



The following example connects two plants, Plant1 and Plant2, in an Interplant network. Both plants procure parts from each other. Plant1's data set resides in */customer/plant1*, while Plant2's data set resides in */customer/plant2*. A *<Tab>* separates each field from adjacent fields. The following files contain the shown data records:

- */customer/plant1/supplier_data* contains:

  Plant1 SELF /customer/plant1
  Plant2 INTERPLANT /customer/plant2

- */customer/plant2/supplier_data* contains:

  Plant1 INTERPLANT /customer/plant1
  Plant2 SELF /customer/plant2

- */customer/plant1/supplier_part_data*

  Along with normal vendor records, this file contains records specifying parts procur-

able from Plant2 via *rhythm_interplant*:
Plant2 Part100 2 WEEKS

- */customer/plant2/supplier_part_data*
  Along with normal vendor records, this file contains records specifying parts procurable from Plant1 via *rhythm_interplant*:
  Plant1 Part200 2 WEEKS

  If Plant1 called Part200 by the name Part200b, the record would instead be:
  Plant1 Part200 2 WEEKS Part200b
- */customer/plant1/spec_file* contains:

```
std_spec_file spec_file;

interplant_data_Plant1
use: interplant_data_PLANTNAME
mode: write;

interplant_data_Plant2
use: interplant_data_PLANTNAME
mode: read;
```

- */customer/plant2/spec_file* contains:

```
std_spec_file spec_file;

interplant_data_Plant2
use: interplant_data_PLANTNAME
mode: write;

interplant_data_Plant1
use: interplant_data_PLANTNAME
mode: read;
```

Appendices

Appendix A

Deploying Without *rhythm_interplant* Executables

Some customers will not be able to use the *rhythm_interplant* executable for some or all of their file transfers, because some or all of their plants may not be connected on a network. Implementing the transfers might involve tape transfers by human operators or modem connections. The following information summarizes how to deploy Interplant in such cases.

The customer may need to develop a procedure (involving human operators, a UNIX or other OS script, or both) which exchanges the files among the plants in the Interplant Network. The primary challenges are:

- Make sure that the files arrive in the correct directories (the data directories of each server).

  As a convenience, it is permitted to send every plant a particular plant's *interplant_data_xxx* file, even to plants which do not procure any parts from it. For instance, if Plant1 procures from Plant2 but not Plant3, it is permitted to send Plant1 the file *interplant_data_Plant3*. It is more robust to send it because, at some future date, the Plant1 data might be changed to procure a part from Plant3.
- If the procedure is run automatically (e.g. through cron jobs), include printouts which show the user whether the data files were exchanged.
- Check that the procedure is run before the *rhythm_server* for each plant is run. Otherwise, each server will be running with old data.
- Check that users know to *Save Plan* at some time before the procedure is run. Otherwise, old versions of Interplant data files will be distributed. No errors will result, but poorer delivery date performance will occur.

If every plant's data is in the same file system, the script will be easy to write (e.g. using the UNIX cp command). Otherwise, the script might require more complicated processes such as rcp, FTP, or possibly a modem transfer mechanism such as Kermit or Z-Modem. Where communications are not implemented, the script can become a procedure where a user transfers data on disk or tape via carrier.

Index